Offline Software

Mark Messier
MIPP @ FNAL
10 August 2002

- Conventions
  - MC Interface
    - Build System/Release management
      - Simple event display
        - To Do ...

Getting Organized...

After last meeting there's been lots of work getting offline effort organized:

- Offline web page setup off MIPP web page
- Regular phone conferences (~every 2 weeks, Fridays at 1 Central)

Agreed to a set of conventions for the offline software:

- C++ based with allowances for FORTRAN wrappers and legacy code where re-coding would be too costly

- ROOT for data format and for other framework services

- SoftwareReleaseTools (SRT) for release management and build system

- PostgreSQL for database

- Adopted set of C++ coding conventions (stolen from MINOS)

- Actually wrote some code (!)...

Offline Package Overview

Currently there are several packages in the repository:

Monte Carlo packages (FORTRAN, Raja)

Monte Carlo interface packages (C/C++, Messier):
- Geant3Interface - C wrappers for G3 subroutines and commons
- E907MCInterface - produces root file from Monte Carlo
- MCClasses - C++ objects for MC hits and particle stack

EventDataModel (C++, Messier)
- Mechanism for putting and retrieving data into event store

EventDisplay (C++, Messier, Johnson)
- RIght now just a very simple display for MC classes

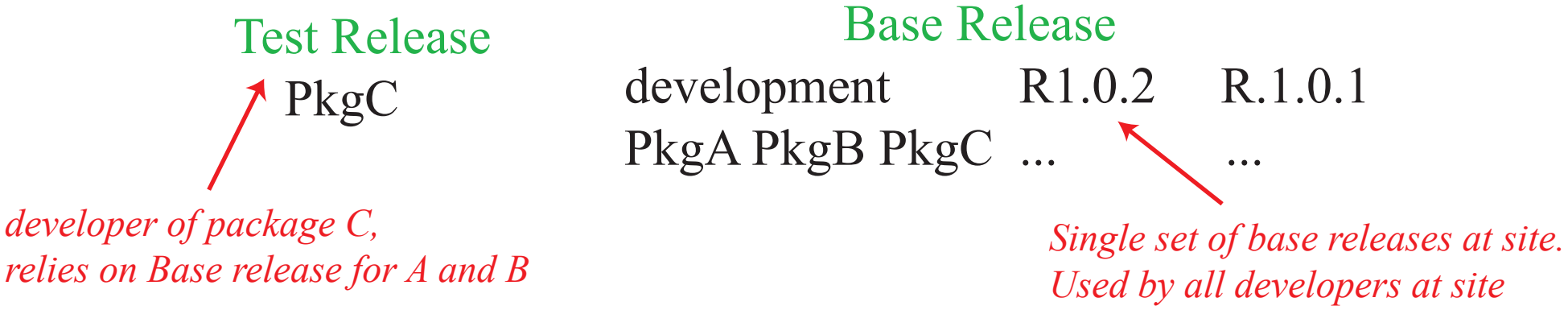
mipp_io (C, Lebedev)
- Input/Ouput package for online data

Software Release Tools

- Provides build system. Simplifies makefiles. Eg.

```
LIB          = lib${PACKAGE}
LIBCXXFILES = *.cxx
include SoftRelTools/arch_spec.mk
```

- Allows developers to work with individual packages:



- Provides framework for making tagged releases

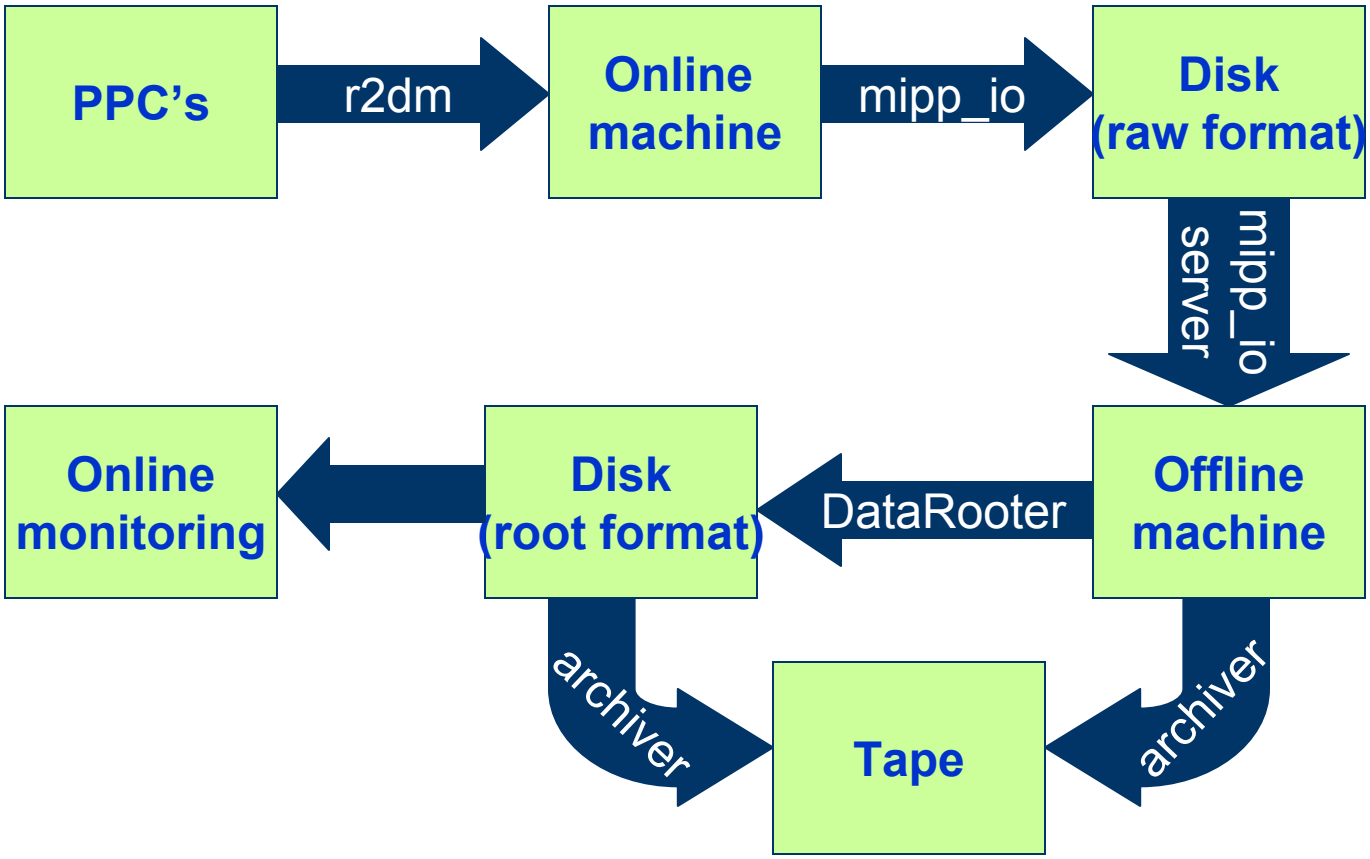
- David Lange has scripts to bootstrap new release of software

- Instructions for SRT and manual on MIPP offline page

Work on Raw Data File

MIPP Collaboration Meeting
August, 2002, FNAL
Andre Lebedev
Harvard University
(presented by Mark Messier)

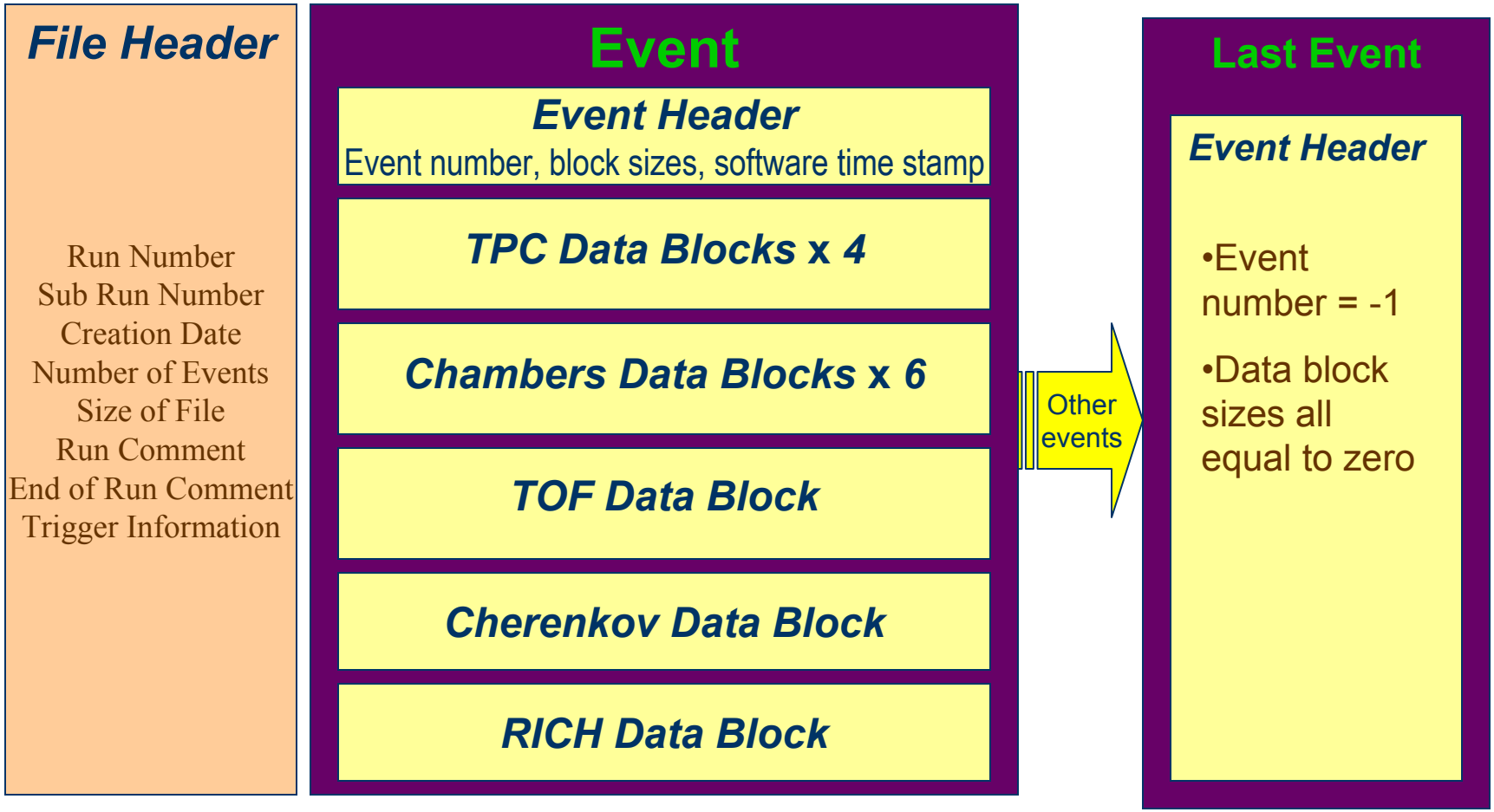
MIPP Data Path



mipp_io Package

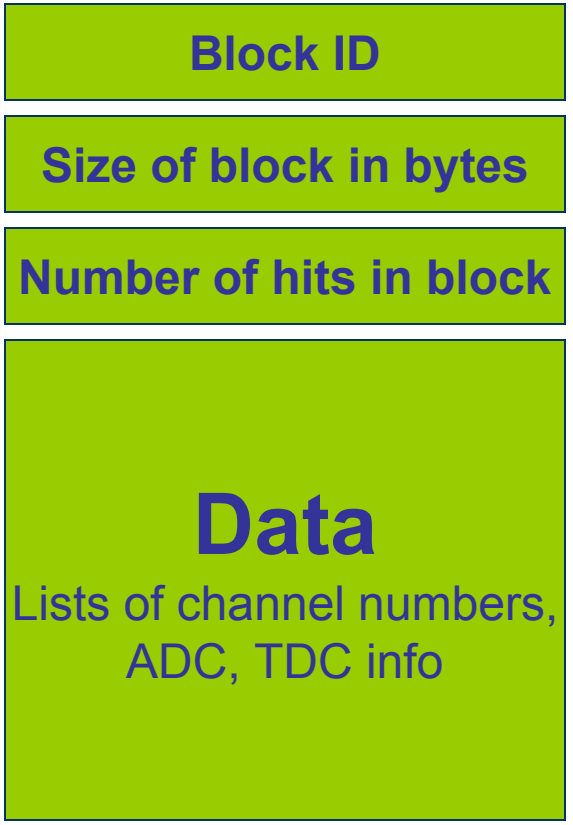
- Defines the layout of the Raw Data File
- Provides for fast writing to disk
  - Writes at speeds of over 100 MB/s
  - Data is physically written in blocks of 131 kB
- Provides a way of serving a data file to DataRooter (a test server application exists)

MIPP Raw File Data format



Stucture of Data Blocks

- Blocks contain info about themselves which is used to decode data from a block, and helps to make sure that data is not corrupted
- Data is packed in bytes, rather than in bits
- People need to tell Andre what kind of information will be stored for various detectors!

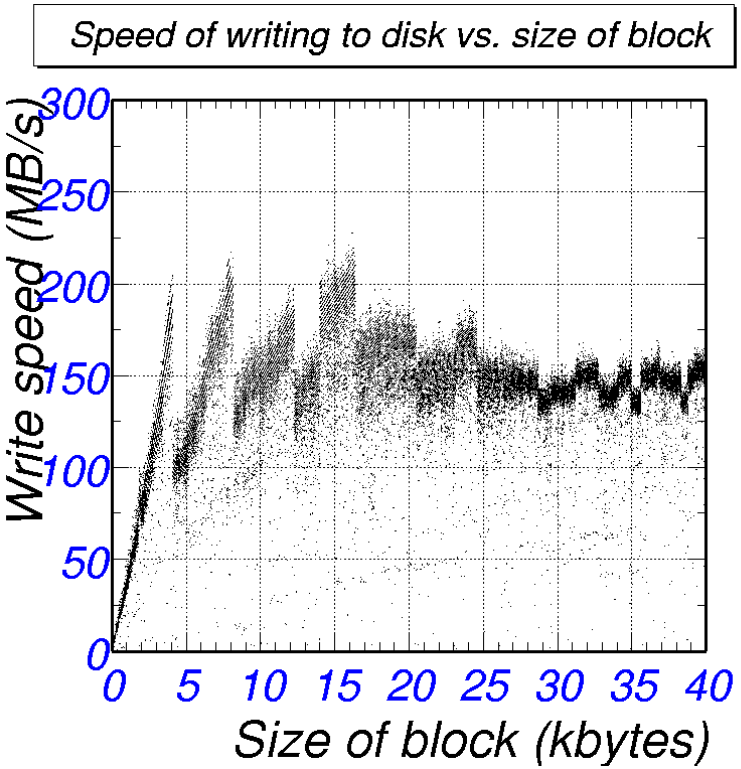


How mipp_io works

- A binary file is created, and file header is written
- When user wants to write an event to disk, the data gets stored in memory until the buffer is big enough or until user flushes it
- Before the file is closed, file header is modified to add end of run comment, number of events, and size of file

Why write data in big chunks?

- It is fastest to write chunks of 4096 bytes (page size on most OS)
- We expect events to be ~100 kB, and will write data to disk in integer number of pages



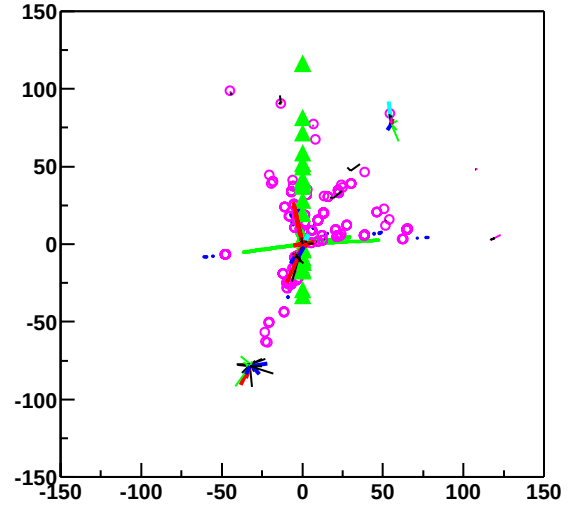
Possible Excerpt from DAQ Code

```
/* Declare structs */
struct mipp_file Mf;
struct mipp_event Mevent;
/* Initialize file for writing */
init_mipp_file(&Mf, file_name, run_number, sub_run,
               trigger_info, MIPP_FWRITE);
/* Add a run comment up to 256 characters */
strcpy(&Mf.comment, "This is run comment");
open_mipp_file(&Mf);
write_file_header(&Mf);
while(/* running condition */) {
  /* Fill mipp_event struct appropriately */
  write_mipp_event(&Mf, &Mevent);
}
/* Add end of run comment up to 256 characters */
strcpy(&Mf.end_of_run_comment, "This is end of run comment");
/* Close file */
close_mipp_file(&Mf);
```

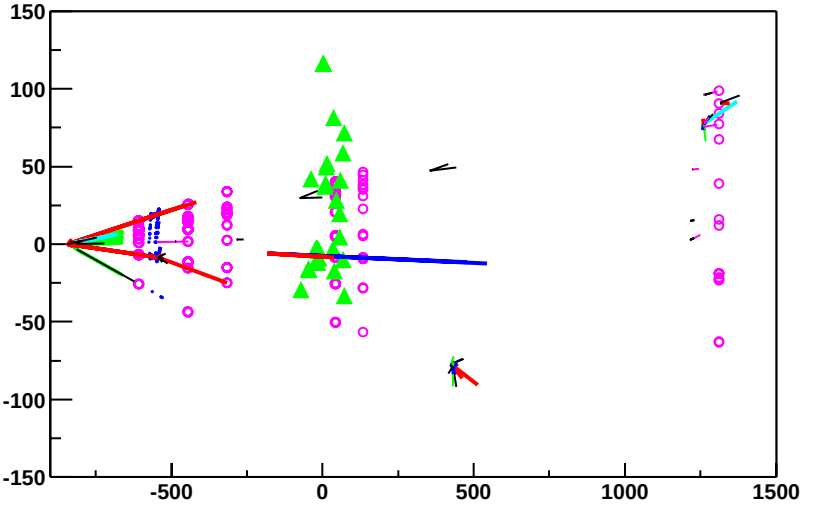
DataRooter

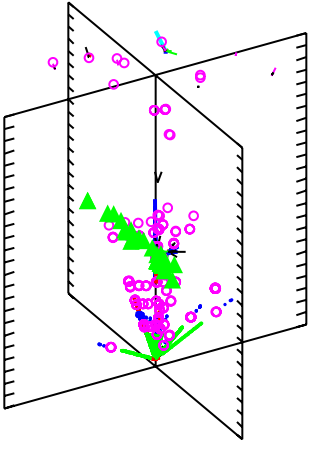
- Eventually, will read data through socket into memory and write it out to disk in root format.
- Inteface between C (online code) and C++ (offline code)
- A test Client application exists which connects to the server and gets data stored in DRClient class
- This is still work in progress

Y vs X

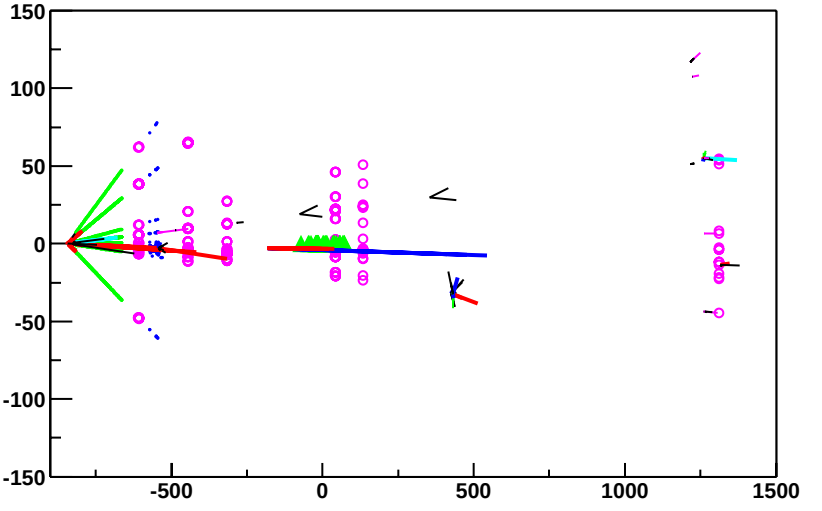


Y vs Z





X vs Z



EventDataModel

- EventData has three parts which can be read independently:
  Header - run/event number, date, etc.
  Data - Monte Carlo and Detector data objects
  Summary - DST-style summary bank
- Can divide Data further if performance improvements are large but for now keep things simple
- Inside "Data" objects are stored in unix-like directory structure:

```
/mc/hits/(hit objects)
/mc/kine/(particle stack)
/raw/(raw data objects)
...
```

- Users can use event store for temporary objects, eg.

```
/tmp/tpc/clusters
/tmp/hists
```

- Eventually will have output module to filter directories on output.
  For example, objects in /tmp would (optionally) not get written out.
- Still need I/O interface to simplify file interface

E910 TPC Reconstruction Code

- Provided code by Brian C.
- E910 used database tables for all data, including data we use event store for
- MIPP vector-based event store makes transition relatively straight forward
- Translate basic objects to ROOT objects, translate tables to vectors.
- Basic syntax ends up staying the same
- Sorted existing code in order of dependency and started translating lowest level objects.
- About 1/6th of the way there...

Database

Offline will use database to store:

- Geometry
- Calibration constants
- Algorithm configurations
- Connection map
- Channel masks
- Run/Beam configuration

Need to provide mechanism to change algorithm parameters interactively

Need way to get Geometry back into MC

Would be nice to have certain tables available as root files for rapid compile/debug cycles

David L. is working on this

Packages looking (maybe found) lead developers

EventDisplay (Steve J.)

Geometry (Jane B-H.)

Online monitoring (??)

needed for engineering run.
MINOS has had good experience using
a CDF framework for this. Interested?
Talk to me...

Need to get raw data format and MC digits in place